AMENDMENTS TO THE SPECIFICATION:

Page 15, amend the sixth full paragraph (lines 14-15) as follows:

Figs. 5A ~~and 5B~~ through 5F are a flow chart showing selected operations performed by the interpreter of Fig. 2.

Page 21, amend the third full paragraph (lines 16-18) as follows:

Interpreter 24 spends most of its time (as does any interpreter) in a loop such as that shown in Figs. 5A ~~and 5B~~ through 5F. Within this loop, interpreter 24 performs a number of operations to support virtual threading.

Page 25, amend the only full paragraph (lines 5-21) as follows:

If the recipient thread is not an inactive or idle thread 42, as determined by interpreter 24 in investigation 84, the interpreter commences a routine 100 (Fig. 5B) for transferring a queued thread 64 from linked list 62 to the associated active thread list 56. Routine 100 is also undertaken by interpreter 24 upon transfer of a recipient thread from the idle thread list 34 to the queued thread list 56 of the least busy native thread 50. In a first step 102 of routine 100, interpreter 24 locks the native thread's queue mutex 52. The interpreter 24 then checks at 104 whether the queued thread list 62 contains at least one virtual thread 64. If so, the first thread in the queue is removed from the queue in a

step 106 and added to the respective active thread list 56 in a step 108. Again, this shifting of a virtual thread from one list to another merely entails an alteration in three pointers, that of the moved thread and those of the immediately preceding threads in the two lists. After shifting of the thread, interpreter 24 decides at a junction 110 whether the newly transferred virtual thread is in a higher priority group than the currently executing thread. If so, a context switch is performed by interpreter 24 and particularly by module 32 (Fig. 3) in a step 112 so that the newly shifted thread becomes the currently executing thread. Queue mutex 52 is then unlocked in a step 114.

Page 26, amend the first full paragraph (lines 2-4) as follows:

Timer processing is not included in the flow chart of Figs. 5A ~~and 5B~~ through 5F as timer processing is not a critical part of virtual threading. An interpreted language will, however, typically need to provide developers with a means of setting timers.

Page 26, amend the third full paragraph (lines 8-14) as follows:

Each time the main interpreter loop (Figs. 5A ~~and 5B~~ through 5F) iterates, usually during event processing, interpreter 24 should check the first entry in the timer list to see if that entry has expired (if it hasn't, no other timer has expired because the timers later in the list expire after the first one in the list). If the timer has expired, it should be removed from the list and an event should be

generated. The event will cause the appropriate virtual thread to be activated (if it's not already active), and that thread will find the timer expiration event when the thread checks its queue.

Page 30, amend the second full paragraph (lines 5-17) as follows:

A simple priority system, embodied in the flow chart of Figs. 5A ~~and 5B~~ <u>through 5F</u>, works by assigning each thread a numerical value for its priority. Typically, this value is between 0 and 100, with larger values indicating higher priorities, but any range of values can be used. In this simple priority system, a given active thread 58 will not be allotted any processor time if there are any higher priority threads in the respective active thread list 56. Each native thread 50 in this system keeps track of the priority level of the highest priority virtual thread 58 in its active list 56 (this priority level can be called the *highest active priority*). When a native thread 50 performs a context switch and must select a new virtual thread 58 to become the current virtual thread, the native thread 50 always selects the next virtual thread in the list 56 that is at the highest active priority, starting over from the beginning of the list when the native thread reaches the end of the list. Each native thread 50 also keeps track of the number of active virtual threads 58 at the highest active priority.

Amend the paragraph bridging pages 33 and 34 as follows:

This procedure is illustrated in ~~Fig. 5A~~ <u>Figs. 5B and 5C</u>. The interpreter 24 first makes a check 118 as to whether, in the respective list 56 of active virtual threads 58, there are any threads of the same priority group as the current

thread. An affirmative outcome leads the interpreter 24 to select the next thread in the current priority group in a step 120 (Fig. 5C). As discussed above, the interpeter 24 then determines, in a step 122, the priority H of the highest priority active virtual thread in the current priority group, the priority P of the selected thread, and the skip count S of the selected thread. At a subsequent decision junction 124, the interpreter 24 inquires whether the skip count S is greater than or equal to the difference between the priority H of the highest priority active virtual thread in the current priority group and the priority P of the selected thread. If the skip count S is greater than or equal to the difference H-P, the interpreter 24 resets the skip counter 116 of the selected thread back to zero in a step 126 and makes a context switch in a step 128. The selected thread has now become the currently executing thread. If the skip count S is less than the difference H-P, the interpreter 24 increments the contents of the skip counter 116 of the selected thread in a step 130 and returns to step 120 to select another thread in the current priority group of the active thread list 56.

Amend the paragraph bridging pages 34 and 35 as follows:

During a time slice of an active virtual thread 58, the interpreter 24, and more particularly instruction execution unit 28, of the native thread 50 to which that virtual thread has been assigned repeatedly reads (step 132) (Fig. 5D) and executes (step 134) instructions from the virtual thread, as quickly as possible. After executing each instruction in step 134, the native thread 50 (i.e., respective instance of the interpreter 24) makes a series of checks 136, 138, 140 (Fig. 5E)

to determine whether the current virtual thread is becoming idle as a result of the instruction, whether the current thread is terminating as a result of the instruction that was just executed, or whether the virtual thread's time slice has expired. If any of these conditions is true, the native thread 50 or respective interpreter 24 stops executing instructions from that virtual thread until that thread becomes active or is assigned a new time slice.

Page 36, amend the first full paragraph (lines 2-11) as follows:

If the native thread 50, i.e., the respective instance of interpreter 24, discovers at check 136 that the currently executing thread has become idle as a result of the last executed instruction thereof, that thread is removed from the respective active thread list 56 in a step 142 (Fig. 5F). This removal requires adjustment in the pointer of the active thread immediately preceding the removed thread in the active thread list 56. The queue mutex 52 of the respective native thread 50 is then locked in a step 144, the newly idled thread is inserted in idle thread list 34 in a step 146, and the mutex 52 is unlocked in a step 148. The insertion of a thread into the idle thread list 34 entails the adjustment of pointers of the inserted thread and of the thread immediately preceding the inserted thread in the idle thread list 34.

Page 36, amend the second full paragraph (lines 12-15) as follows:

If the native thread 50, i.e., the respective instance of interpreter 24, discovers at check 138 that the currently executing thread has terminated its task or job as a result of the last executed instruction, the thread is removed from

active list 56 in a step 150 <u>(Fig. 5F)</u>. Also, resources are freed which were being used by the terminated thread.

Amend the paragraph bridging pages 36 and 37 as follows:

After the transfer of a thread from active list 56 to idle list 34 in steps 142, 144, 146, 148 <u>(Fig. 5F)</u> or after the removal of a terminated thread from active list 56 in step 150, the interpreter 24 of the relevant native thread 50 investigates at 152 whether there are any threads left in the active list 56 in the same priority group as the current thread. If so, the interpreter 24 returns to perform maintenance tasks in step 70 (Fig. 5A). If not, the interpreter 24 or native thread 50 runs through the active list 56 in a step 154 to determine the priority of the highest priority thread in the list. In subsequent steps 156 and 158, the highest priority thread is made the current thread and the number of threads in the active list 56 at the highest priority level is counted. The interpreter 24 returns to perform maintenance tasks in step 70.

Page 37, amend the first full paragraph (lines 4-17) as follows:

After having determined at checks 136 and 138 <u>(Fig. 5E)</u> that the current thread has not become idle or terminated its task upon the execution of a last bytecode instruction, interpreter 24 queries at a decision junction 160 whether the current thread's priority group has changed as a result of the instruction that was just executed. If not, check 140 is undertaken to determine whether the time slice has expired. If so, interpreter 24 questions at 162 <u>(Fig. 5D)</u> whether the

priority group of the current thread has increased or decreased. In the case of an increase, the thread number in the highest priority group is reset to unity in a step 164. In the case of a decrease, the count of threads in the highest priority group is decremented in a step 166. The interpreter 24 investigates at a decision junction 168 whether there are any active threads left in the highest priority group. A negative outcome to this investigation leads the interpreter 24 to scan the active threads in a step 170 to determine a new highest priority group and to count the number of threads in that group. The interpeter 24 then undertakes a context switch in a step 172, with the highest priority active thread 58 becoming the current thread.

Page 37, amend the second full paragraph (lines 18-20) as follows:

A positive outcome to investigation or decision junction 168 <u>(Fig. 5D)</u> leads the interpreter 24 directly to context switch step 172 and from that [[sep]] <u>step</u> to read and execute steps 132 and 134.